

Sun Microsystems, Inc.  
Intellectual Property Law Department  
901 San Antonio Road, MS UPAL1-521  
Palo Alto, CA 94303  
(650) 786-7852  
(650) 786-2325



RECEIVED  
MAY 22 2001  
Technology Center 2100

# 2122  
Dwyatt  
6-21-01

Docket No.: P4756 US

May 15, 2001

Box NO FEE  
Commissioner For Patents  
Washington, D.C. 20231

RE: Applicant(s): Foster, Tim S.; Lahiff, Louise Christine  
Assignee: Sun Microsystems, Inc.  
Title: SOFTWARE PACKAGE VERIFICATION  
Serial No.: 09/776,328 File Date: February 2, 2001  
Examiner: To Be Assigned Group Art Unit: 2122

Dear Sir:

Transmitted herewith are the following documents in the above-identified application:

1. Return Receipt Postcard;
2. This Transmittal Letter (1 page, in duplicate); and
3. Certified Copy of Priority Document: European Application No. 00305775.9 filed on July 7, 2000.

- ☒ No additional fee is required.  
☐ The fee has been calculated as shown below:

Total fee for this submission:

\$ 00.00

- ☒ The Commissioner is hereby authorized to charge any additional fees required and credit any overpayment to our Deposit account No.: 19-4974.

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Assistant Commissioner For Patents, Washington, D.C. 20231, on May 15, 2001.

Attorney For Applicant(s)

Date of Signature

Respectfully submitted,

Elaine K. Lee  
Attorney for Applicant(s)  
Reg. No.: 41,936

**THIS PAGE BLANK (USPTO)**



Europäisches  
Patentamt

European  
Patent Office

Office européen  
des brevets



Bescheinigung

Certificate

Attestation

RECEIVED  
MAY 22 2001  
Technology Center 2100

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

00305775.9

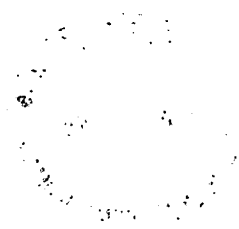
Der Präsident des Europäischen Patentamts;  
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets  
p.o.

I.L.C. HATTEN-HECKMAN

DEN HAAG, DEN  
THE HAGUE, 28/03/01  
LA HAYE, LE



**THIS PAGE BLANK (USPTO)**



Europäisches  
Patentamt

European  
Patent Office

Office européen  
des brevets

**Blatt 2 der Bescheinigung**  
**Sheet 2 of the certificate**  
**Page 2 de l'attestation**

Anmeldung Nr.:  
Application no.:  
Demande n°: 00305775.9

Anmeldetag:  
Date of filing:  
Date de dépôt: 07/07/00

Anmelder:  
Applicant(s):  
Demandeur(s):  
SUN MICROSYSTEMS, INC.  
Palo Alto, California 94303  
UNITED STATES OF AMERICA

Bezeichnung der Erfindung:  
Title of the invention:  
Titre de l'invention:  
Software package verification

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat:  
State:  
Pays:

Tag:  
Date:  
Date:

Aktenzeichen:  
File no.  
Numéro de dépôt:

Internationale Patentklassifikation:  
International Patent classification:  
Classification internationale des brevets:

G06F11/36

Am Anmeldetag benannte Vertragsstaaten:  
Contracting states designated at date of filing: AT/BE/CH/CY/DE/DK/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/PT/SE  
Etats contractants désignés lors du dépôt:

Bemerkungen:  
Remarks:  
Remarques:

**THIS PAGE BLANK (USPTO)**

SUN REF: P4756EP  
DYC REF: P008727EP

1

## BACKGROUND OF THE INVENTION

The invention relates to the checking of software packages with a view to improving software package quality.

5

As software installation packages (which could also be termed software delivery packages, but for reasons of conciseness are referred to as software packages hereinafter), become more and more complex, the task of verifying the structure and content of such packages becomes more and more difficult. A software package as  
10 supplied for example, on a medium such as disk, or over a network can involve many program and data components (hereinafter software components), such as data files, program files, program modules, etc. Typically, the process of verifying that the software components are valid in themselves, are consistent with each other, and are consistent with a target platform or system has involved manual checking of those  
15 components by the members of a development team and/or an associated quality control group.

There are individual tools that can provide isolated checks on a software package. For example, a program called "pkgchk" is available on the Solaris operating system  
20 platform (Solaris is a Trademark of Sun Microsystems, Inc) is able to provide a check of package structure integrity. However, even with such tools, manual co-ordination of tests to be performed is required, which is expensive, time consuming, and prone to human error.

25 An aim of the invention is, therefore, to enable more effective and reliable verification of software packages in a flexible manner.

## SUMMARY OF THE INVENTION

Particular and preferred aspects of the invention are set out in the accompanying independent and dependent claims. Combinations of features from the dependent  
5 claims may be combined with features of the independent claims as appropriate and not merely as explicitly set out in the claims.

An aspect of the invention provides a software package verification tool for verifying a software package that includes at least one software component. The tool includes a  
10 framework operable to identify at least one test module defining a test of at least one parameter of a software component of the package. It also includes a control module operable to access the framework to cause at least one test module identified thereby to perform the test defined thereby for verifying the package

15 By providing a framework within which individual test components may be added or deleted as required, a flexible test structure can be provided for software packages.

Typically, the framework will identify a plurality of test modules for verifying the correctness of a particular software package. In such a case, the framework preferably  
20 identifies a priority for each test module, that is, it identifies the order in which tests are to be performed. This enables the ordering the tests to be efficient, and would avoid taking time carrying out tests that might in any case be redundant if, for example, the software package failed a fundamental test.

25 Using the priority information, the control module can be operable sequentially to cause the test modules to be operable according to the test module priority identified in the framework.

A mechanism can be provided for identifying test modules that are active and test  
30 modules that are not active (i.e., which test modules are to be used for performing



tests, and which are not to be used). This could, for example, be provided by means of the framework, or by means of the control module.

In one example of the invention, the framework comprises a directory having a plurality of entries, each entry identifying a test module. Each entry can define a priority of the test module identified thereby. Alternatively, or in addition, the identity (e.g. the file name) of a test module can define its priority. Each entry can also include an indicator as to whether the test module is to be active or not for a particular sequence of tests.

10

Examples of tests of software package parameters that may be performed by respective test modules are as follows::

- a test of package structure integrity;
- a test that all components are compiled using a compatible compiler version;
- 15 a test that all binaries are for the same architecture;
- a test that a copyright module is current;
- a test that only specified modules are present;
- a test of changes with respect to a previous version of the package.

20 An embodiment of the invention also includes at least one test module. Each test module can be formed by a script and the framework can identify a test module by a name for the script for that module. They could alternatively be formed by software objects.

25 The tool can be in the form of a computer program, for example in the form of computer program instructions provided on a carrier medium. The carrier medium could, for example, be a storage or a transmission medium.

Another aspect of the invention provides a system comprising a mechanism for  
30 verifying a software package that includes at least one software component. The

system includes a framework operable to identify at least one test module defining a test of at least one parameter of a software component of the package and a control module operable to access the framework for causing at least one test module to perform the test defined thereby for verifying the package.

5

The system can include a computer with a processor, memory and software held in the memory and operable to control the processor. The software can form a framework operable to identify at least one test module defining a test of at least one parameter of a software component of the package and a control module. The control module can  
10 be operable to access the framework to cause at least one test module identified thereby to perform the test defined thereby for verifying the package.

A further aspect of the invention provides a method of verifying a software package that includes at least one software component. The method includes providing a  
15 framework for identifying at least one test module, each said test module defining a test of at least one parameter of a software component of the package. It further includes accessing the framework to identify at least one test module and causing the test module to perform the test defined thereby on the package. The method can further include reporting test results.

## BRIEF DESCRIPTION OF THE DRAWINGS

Exemplary embodiments of the present invention will be described hereinafter, by way of example only, with reference to the accompanying drawings in which like  
5 reference signs relate to like elements and in which:

Figure 1 is a schematic representation of a computer workstation for an exemplary implementation of the invention;

10 Figure 2 is schematic block diagram illustrating an exemplary configuration of a computer workstation as shown in Figure 1;

Figure 3 is schematic representation of a software hierarchy of an exemplary implementation of the invention;

15

Figure 4 further represents the relationship between elements of an exemplary implementation of the invention;

Figure 5 illustrates a directory forming part of an exemplary implementation of the  
20 invention;

Figure 6 is a schematic representation of a software package to be verified by an embodiment of a software package verification tool according to the invention;

25 Figure 7 is a flow diagram illustrating the operation of an embodiment of a software package verification tool according to the invention.

## DESCRIPTION OF PARTICULAR EMBODIMENTS

Exemplary embodiments of the present invention are described in the following with reference to the accompanying drawings. An embodiment of the invention will be described that is implemented on a single computer workstation, by way of example only. However, the invention can equally be implemented on a host computer accessible by a terminal, on a multiple processor system or over a network comprising a plurality of computer stations. Also, as described later, the invention could be, for example, integrated into an integrated circuit.

10

Figure 1 is a schematic representation of a computer workstation on which an exemplary embodiment of the invention is implemented. As shown in Figure 1, a computer workstation 10 includes a system unit 12 that includes a processor, memory, etc (see Figure 2), user input devices, for example in the form of a keyboard 14 and a mouse 16, and a display 18. Removable media devices in the form, for example of a floppy disk drive 20 and an optical and/or magneto-optical drive (e.g. a CD, a DVD ROM, a CDR drive) 22 can also be provided.

Figure 2 is schematic block diagram illustrating an exemplary configuration of a computer workstation 10 as shown in Figure 1.

As shown in Figure 2, the computer workstation 10 includes a bus 30 to which a number of units are connected. A microprocessor (CPU) 32 is connected to the bus 30. Main memory 34 for holding computer programs and data is also connected to the bus 30 and is accessible to the processor. A display adapter 36 connects the display 18 to the bus 30. A communications interface 38, for example a network interface and/or a telephonic interface such as a modem, ISDN or optical interface, enables the computer workstation 10 to be connected 40 to other computers via, for example, an intranet or the Internet. An input device interface 42 connects one or more input devices, for example the keyboard 14 and the mouse 16, to the bus 30. A floppy drive

interface 44 provides access to the floppy disk drive 20. An optical drive interface 46 provides access to the optical or magneto-optical drive 22. A storage interface 48 enables access to a hard disk 50. Further interfaces, not shown, for example for connection of a printer (not shown), may also be provided. Indeed, it will be appreciated that one or more of the components illustrated in Figure 2 may be omitted and/or additional components may be provided, as required for a particular implementation.

Figure 3 is a schematic representation of a software hierarchy of an exemplary embodiment of the present invention. The software hierarchy 60 illustrates the operating system (O/S) 62 of the computer workstation 10. A software package verification tool 64 includes a framework 68 that enables a control module 66 to access test modules 70 for testing the operation of a software package 72. Also shown in Figure 3 is a previous version 72' of the software package 72.

15

Figure 4 illustrates the relationship of the various components of the verification tool 64. Thus, the control module 66 is operable to access the framework 68 that in turn provides access to the test modules 70. Each of the test modules 70 is a test routine that, on initialisation, is registered with the framework 68, whereby the framework 68 is able to identify the test modules 70 for use in testing the software package 72. The registration can be effected using standard operating system techniques.

As illustrated in Figure 5, in one embodiment of the invention, the framework 68 is operable to provide a directory 80 that includes a plurality of entries 74. Each entry 74 includes a link (L) 76 to a respective test module. In a particular example shown in Figure 5, each entry 74 is also provided with a priority indicator (P) 78 indicating relative priorities of the test modules 70 identified by the respective links L 76. The priorities P indicate the order in which the various tests performed by the test modules are to be carried out.

30

Although in the particular example shown, a priority indication P is included in each entry 74 in the table, in other examples and embodiments the priority could be indicated in another manner. For example, the priority could be indicated by the link to the respective modules. For example, in an associative table, the link could be identified by the name, or label, applied to each of the modules, and this name could include a sequence number. Thus, for example, the test modules could have names such as TM0, TM1, TM2, etc. The priority order could be such that the lowest number indicates the highest priority, or alternatively that the highest number indicates the highest priority. As a further alternative, the directory 80 could be organised as a linked list, with the position in the list identifying the priority order. The use of separate test modules 70 that are registered with a framework 68, provides a flexible structure in that test modules can be added and/or deleted as required for any particular implementation. Also, particularly where the priority order is indicated by an entry in the directory, or by the organisation of the directory (eg, as a linked list), the order of the test can readily be changed.

Figure 5 also illustrates a further field 75 for each of the entries 74. This further field contains an active indicator (A) 75 that indicates whether the test module relating to that entry is to be active or non-active for a particular sequence of tests. As an alternative to providing such an indication in the framework 68, the control module 66 could be coded to identify the modules to be effected for a particular series of tests. The sequence of test modules to be active could depend, for example, on whether a software package to be verified is a new package, or whether it is a modified package and tests are to be performed comparing the current software package to a previous version of that package.

Figure 6 illustrates an exemplary software package, to be tested by a software verification tool in accordance with the present invention. The software package could be generated within the computer workstation 10 as a result of the work by a user of that station, or could be supplied on a removable medium such as a floppy disk

- or an optical, magnetic or magneto-optical disk, tape, etc, or could be supplied from a remote location via a network (e.g. intranet or internet). In one example of a package, for example as in a software package for the Solaris operating system environment (System 5, release 4 packaging standard), the package 72 includes a file list 82 which
- 5 identifies a plurality of files 84 that make up the package. In the Solaris environment, this is known as the "pkgmap" (package map file). The file list 82 includes a plurality of entries 86. Each entry 86 relates to a respective file 84, and includes data (D) 88 about the file, and a reference (R) 90 to the file 84 concerned. The files 84 can be data and/or program files. The data D in respect of those files can include parameters such
- 10 as the file name, a version number, an operating system environment for which the package is intended, the size of the file, binary data types for the file, etc. It should be noted, however, that the invention is not limited to use with such software packages, and that these might include such a pkgmap file.
- 15 Various of the test modules 70 can be operable to use the data D 88 in respect of the files 84 of the software package 72 to verify the validity and/or correctness of the actual contents of the files 84 compared to the data D 88 in respect thereof. In some cases, the test can be operable to provide an absolute test comparing the data D 88 to the content of the corresponding file 84 and/or to compare a current software package
- 20 version to a previous software package version stored in the memory 34 of the computer workstation 10. For example, data in respect of a prior version of a software package 72 could be stored as 72' in the file hierarchy 60 as shown in Figure 3.
- 25 Figure 7 is a flow diagram illustrating the operation of an embodiment of the invention. This flow diagram sets out the steps that are performed by the software verification tool in response to receipt of a checker command for checking a software package.

On receipt of the checker command 92, the software package verification tool performs initial verification operations in step S1. Step S1 is performed by the control module 66. The initial verification involves checking to see that a checker command C for initialising the checking process contains appropriate arguments for initialising  
5 the checking operation. The checker command 92 can include three or four arguments including an optional field H for optionally hiding details, and indication N as to whether the package is a new package, a new package name NP including the path to that package, and where a new package is to be compared to an old package, the name of the old package OP and the path to the old package.

10

In the initial verification step S1, a check is made to see that there are the correct number of arguments in the checker command 92. A check is made to see that the package (or packages) exist in the directories identified in the paths to those packages, a check to see that the user has permissions on the package(s) concerned and a check  
15 that the package contains actual files and not merely links that refer to a location that would not exist outside a software developer's environment, as well as a final, priority 1 check to see if a package map (pkgmap) and package information (pkginfo) exist for the package(s) concerned. In each case, if one of these tests fails, in then in step S2 control passes to step S8 and an output error message of an appropriate type is issued.  
20 This can be displayed, for example, on the screen 18 of the computer workstation 10 or could be printed in an error log, or could be transmitted to a remote station.

If the initial verification tests in step S1 are positive, then control passes via step S2 to step S3.

25

In step S3, the test module with the next highest priority (for this first time at step S3 the test module with the "next highest priority " is the test module with the highest priority). The priority is selected using the information in the directory 80, if a structure as shown in Figure 5 is provided in the particular embodiment of the  
30 invention, or the priority is selected using any other appropriate technique as



described above. The software package is then tested with the selected test module in step S4. If the software package fails the test in step S4, then control passes via step S5 to step S8 at which point an appropriate error message is output. Alternatively, if the test performed in step S4 is passed by the software package, then control passes  
5 via step S5 to step S6. At this point, with reference to the directory or other structure in the framework for identifying the next test module, a test is performed as to see whether the last test module has now been employed. If there are still test modules to be employed, then control passes to step S3 and the test module with the next highest priority is selected.

10

If, however, the last module had already been tested, then control passes from step S6 to step S7 and a message is output indicating that the software package passed all the tests, and accordingly that the software package can be used and or/delivered to a customer.

15

It will be appreciated that the flow diagram illustrated in Figure 7 is merely one possible flow organisation for implementing an embodiment of the present invention. For example, the initial verification, rather than being performed by the control module, could be provided by a specific test module that always has the highest  
20 priority. Also, the initial verification may be quite different if, for example, the software packages do not include a pkgmap file. Moreover, the detail of the control loop in the flow of Figure 7 could be implemented in different ways. For example, depending on the programming language used, step S6 could effectively be part of step S3. However, it will be noted that individual tests are performed as defined by  
25 separate test modules and that the order of operation of the test modules can be selected by according an appropriate priority to each of those tests. By correctly ordering the tests, it is possible to provide efficient verification, whereby one or more fundamental tests may be performed initially, avoiding the need for subsequent tests to be performed if the fundamental test or tests are not passed by the software

package. The use of a flexible framework as described enables individual test modules to be added or deleted as appropriate.

In the following, a number of examples of possible examples of test modules 70 for  
5 testing different software package parameters will be described. It will be appreciated that these are merely examples of possible test modules, and that other test modules may be envisioned. Also, in any particular embodiment, the specific test modules to be chosen will depend on the requirements of that embodiment. Each of the test modules is, in a particular embodiment of the invention, implemented by a respective  
10 script in a scripting language. However the test modules could be configured as objects in an object orientated language, or by any other form of program module in any language as required for a particular application.

One test module 70 can be to check that a package does not contain any files of zero  
15 size. In order to perform this, the test module script, an initial check to see if the package is compressed. If the package is compressed, it is decompressed and then a check is made to test whether there are any empty files.

Another further module 70 can check that there are no rubbish files in the package,  
20 that is files that were in the package but are not named in the pkgmap file. In this case the test module script compares each file in the package to the content of the pkgmap file. The package fails this test if any file is not referenced in the pkgmap file.

A further test module 70 can test that a correct form of a current copyright notice has  
25 been included in the file. In this the test module script checks that a file contains the current year as the copyright notice.

A test module 70 can check that only specified modules are present.

A further test module 70 can be employed where a new package is compared to a previous package to see if any changes have been effected between packages. In this case the test module script compares the packages to see whether there have been any changes between the current and previous versions of the packages, and to see that the  
5 pkgmap file correctly reflects this.

A further test module can be provided to test whether a software package to be verified is missing files deleted with respect to an earlier version of that package. In this case, the test module script carries out an appropriate comparison of the current  
10 and previous versions of the package.

Another test module can be provided for testing package structure identity, including checking the binary data type, the compiler version that was used to build the binary files, or if the files are sample source code, that they compile clearly against a  
15 particular version of the computer. The binary data type check can be to check whether, for example, 32 or 64 bit data are used. If there is a difference, there can be an architectural problem with the construction of the file. In the context of the compiler version check, a check can be made that the files have been generated by an appropriate compiler. If inconsistent compiler versions are identified, an appropriate  
20 error message can then be output. Also, if, for example, files are compiled using a particular version of a compiler, then a check can be made to ensure that all files were compiled using the same version of that compiler. These tests could be provided by a single test module script, or could be provided by respective test module scripts in respective test modules.

25

Thus, for example, a test that all components are compiled using a compatible compiler version could be performed by a separate test module script in a separate test module 70.

It will be appreciated that one or more of the tests could be combined in common test modules, or each of the tests described above could be implemented using a separate test module. It will further be understood that other test modules can be employed for testing any appropriate parameters for confirming the correctness of a software  
5 package.

There has been described, a software package verification tool that enables verifying a software package that includes at least one software component. The tool includes a framework operable to identify at least one test module defining a test of at least one  
10 parameter of a software component of the package. It also includes a control module operable to access the framework to cause at least one test module identified thereby to perform the test defined thereby for verifying the package.

The framework, within which individual test modules may be added or deleted as  
15 required, provides a flexible test structure for software packages. Typically, the framework identifies a plurality of test modules for verifying the correctness of a particular software package. In such a case, the framework can identify a priority for each test module for effecting an ordering of the tests. This enables the performance of the tests to be efficient, avoiding, for example, unnecessary tests that are redundant  
20 if the software package fails a more fundamental test.

The control module, the framework and the test modules can each be implemented by program code, for example by respective scripts in respective software modules. In one embodiment of the invention each test module is formed by a script and the  
25 framework identifies a test module by a name for the script for that module. However, in other embodiments, other structures and implementations could be employed. For example, software verification tool and/or the test module logic could be embodied in a special purpose integrated circuit such as an application specific integrated circuit (ASIC).

The software tool could be provided as a computer program product, for example on a carrier medium such as a storage medium or a transmission medium.

A computer program product for implementing the invention can be in the form of a  
5 computer program on a carrier medium. The carrier medium could be a storage medium, such as solid state magnetic optical, magneto-optical or other storage medium. The carrier medium could be a transmission medium such as broadcast, telephonic, computer network, wired, wireless, electrical, electromagnetic, optical or indeed any other transmission medium.

10

Examples of tests that may be performed by respective test modules are as follows:

a test of package structure integrity;

a test that all components are compiled using a compatible compiler version;

a test that all binaries are for the same architecture;

15 a test that a copyright module is current;

a test that only specified modules are present;

a test for changes with respect to a previous version of the package.

Although particular embodiments of the invention have been described, it will be  
20 appreciated that many modifications/additions and/or substitutions may be made within the scope of the invention as defined in the appended claims.

For example, tests other than those described herein may be implemented by the test modules. Also, one or more tests could be performed by any one test module, as  
25 appropriate for a particular application or group of applications.

**THIS PAGE BLANK (USPTO)**

## CLAIMS

1. A software package verification tool for verifying a software package that includes at least one software component, the tool comprising:
  - 5 a) a framework operable to identify at least one test module defining a test of at least one parameter of a software component of the package; and
  - b) a control module operable to access a framework to cause at least one test module identified thereby to perform the test defined thereby for verifying the package
- 10 2. The tool of claim 1, wherein the framework identifies a plurality of the test modules.
3. The tool of claim 2, wherein the framework identifies a priority for each test module.
- 15 4. The tool of claim 3, wherein the control module is operable sequentially to cause the test modules to be operable according to the test module priority identified in the framework.
- 20 5. The tool of any of claims 2 to 4, wherein a mechanism is provided for identifying test modules that are active and test modules that are not active.
6. The tool of claim 5, wherein the framework includes the mechanism for
- 25 identifying test modules that are active and test modules that are not active.
7. The tool of claim 5, wherein the control module includes the mechanism for identifying test modules that are active and test modules that are not active.

8. The tool of any of claims 2 to 7, wherein the framework comprises a directory having a plurality of entries, each entry identifying a test module.
9. The tool of claim 8, wherein each entry defines a priority of the test module identified thereby.
10. The tool of claim 8, wherein the identity of a test module defines its priority.
11. The tool of any preceding claim including at least one test module.
12. The tool of claim 11, wherein each test module is formed by a script and the framework identifies a test module by a name for the script for that module.
13. The tool of claim 11, wherein each test module is formed by a software object.
14. The tool of any preceding claim, comprising computer program instructions on a carrier medium.
15. A computer program comprising computer executable instructions for verifying a software package that includes at least one software component, the computer program comprising computer executable instructions:
- a) forming a framework operable to identify at least one test module defining a test of at least one parameter of a software component of the package; and
  - b) forming a control module operable to access the framework to cause at least one test module identified thereby to perform the test defined thereby for verifying the package.
16. The computer program of claim 15 including at least one test module.
17. The computer program of claim 15 or claim 16 on a carrier medium.



18. A system for comprising a mechanism for verifying a software package that includes at least one software component, the system comprising:
- 5 a) a framework to identify at least one test module defining a test of at least one parameter of a software component of the package; and
- b) a control module operable to access the framework for causing at least one test module identified thereby to perform the test defined thereby for verifying the package.
- 10 19. The system of claim 18, wherein the system comprises a computer including a processor, memory and software held in the memory and operable to control the processor, the software forming:
- said framework and said control module.
- 15 20. The system of claim 18 or claim 19, comprising at least one test module.
21. A method of verifying a software package that includes at least one software component, the method comprising:
- a) providing a framework for identifying at least one test module, each said test
- 20 module defining a test of at least one parameter of a software component of the package;
- b) accessing the framework to identify at least one test module; and
- c) causing the test module to perform the test defined thereby on the package.
- 25 22. The method of claim 21, wherein the framework identifying a plurality of the test modules.
23. The method of claim 22, wherein a priority for each test module is identified in the framework.

24. The method of claim 22, comprising sequentially causing the test modules to be operable according to the identified test module priority.
25. The method of claim 22, comprising identifying test modules that are active  
5 and test modules that are not active.
26. The method of claim 25, comprising identifying in the framework test modules that are active and test modules that are not active.
- 10 27. The method of claim 25, comprising identifying in the control module test modules that are active and test modules that are not active.
28. The method of claim 22, comprising providing a directory in the framework,  
wherein the directory has a plurality of entries and each entry identifies a test  
15 module.
29. The method of claim 28, wherein each entry defines a priority of the test module identified thereby.
- 20 30. The method of claim 28, wherein the identity of a module defines its priority.
31. A method of verifying a software package that includes at least one software component, the method comprising:
- a) receiving the software package;
- 25 b) accessing a framework that references at least one test module, each said test module defines a test of at least one parameter of a software component of the package, for identifying at least one test module from the framework; and
- c) performing the test defined by the test module on the package.

32. The method of claim 31, including repeating steps (b) and (c) to perform a sequence of tests, the order in which the tests are performed being determined by relative priorities assigned to the test modules.

**THIS PAGE BLANK (USPTO)**

## ABSTRACT

SOFTWARE PACKAGE VERIFICATION

5

A software package verification tool enables verifying a software package that includes at least one software component. The tool includes at least one test module defining a test of at least one parameter of a software component of the package. It also includes a control module operable to access a framework that identifies each test module and to cause at least one test module to perform the test defined thereby for verifying the package. The framework, within which individual test modules may be added or deleted as required, provides a flexible test structure for software packages. Typically, the framework identifies a plurality of test modules for verifying the correctness of a particular software package. In such a case, the framework can identify a priority for each test module for effecting an ordering of the tests. This enables the performance of the tests to be efficient, avoiding, for example, unnecessary tests that are redundant if the software package fails a more fundamental test.

20 Fig 4 for Abstract

**THIS PAGE BLANK (USPTO)**

1/7

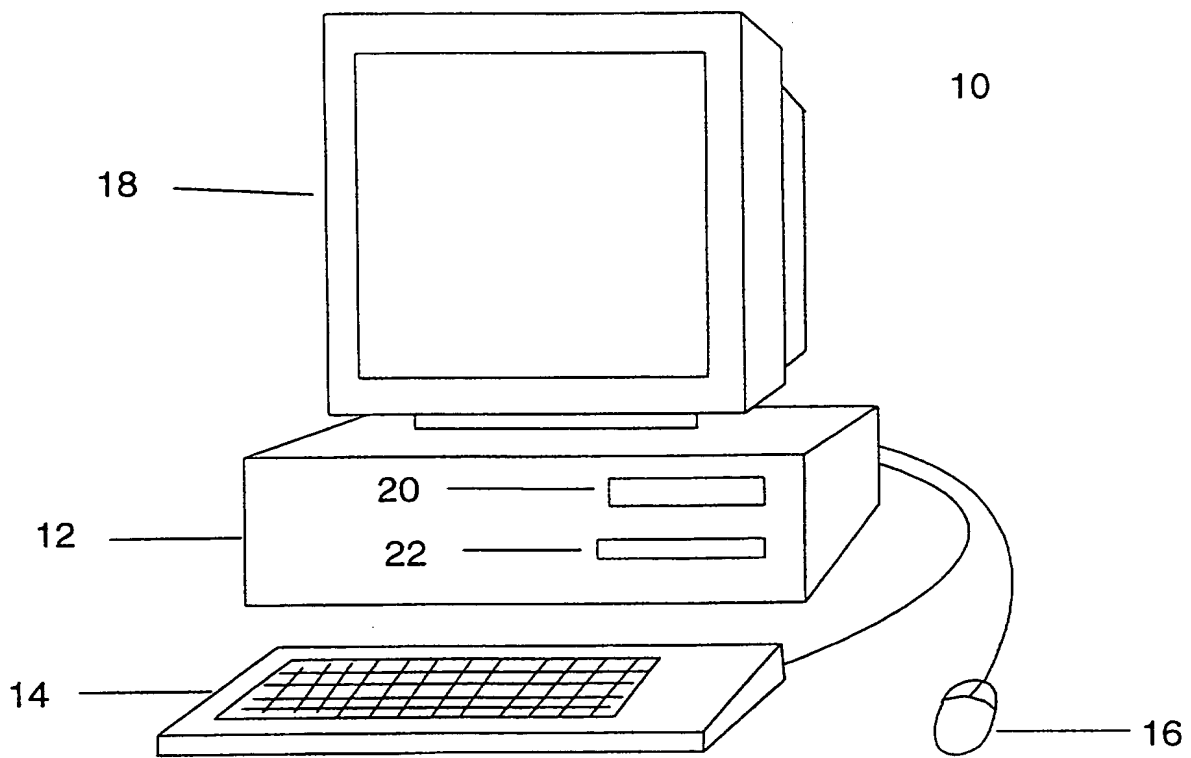


Fig. 1

2/7

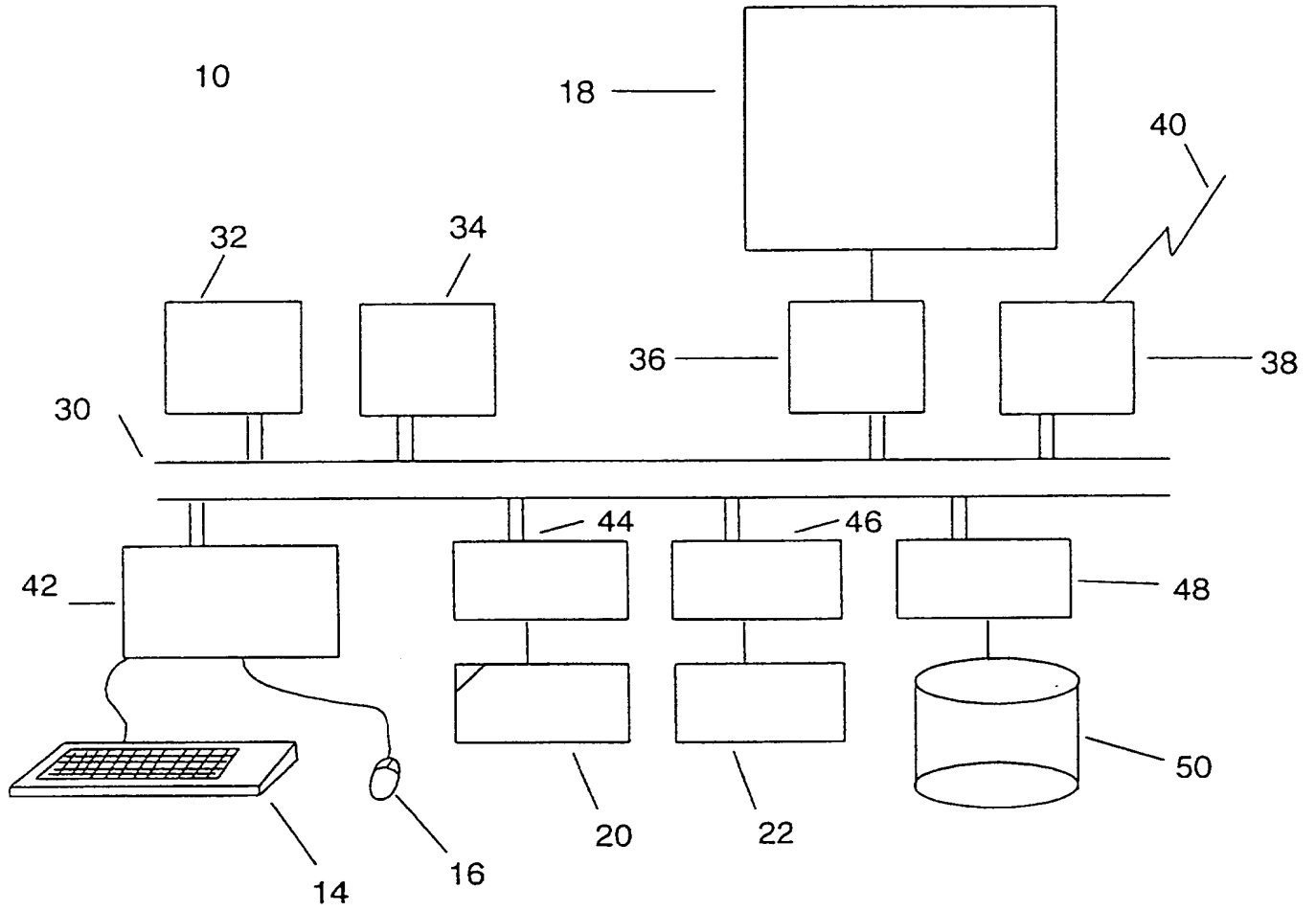


Fig. 2



3/7

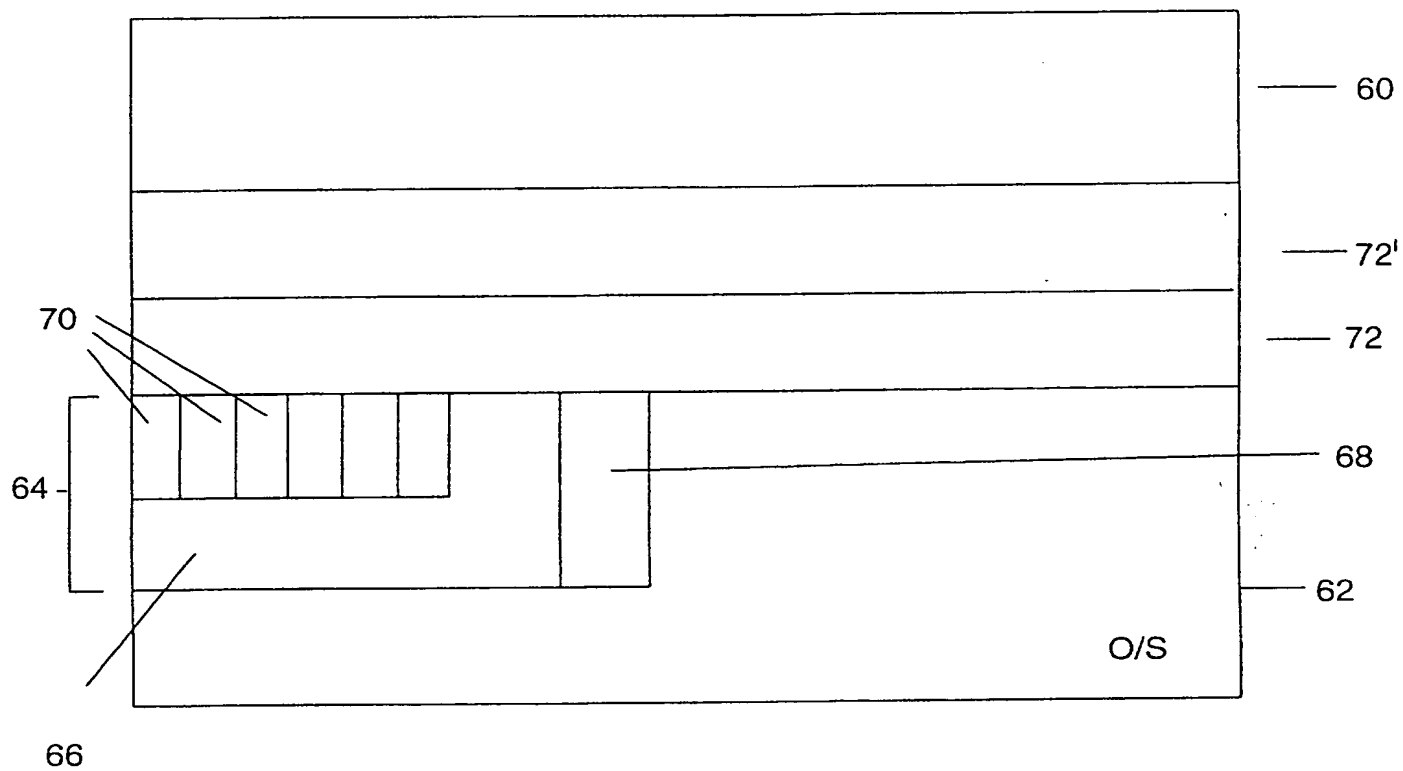


Fig.3

4/7

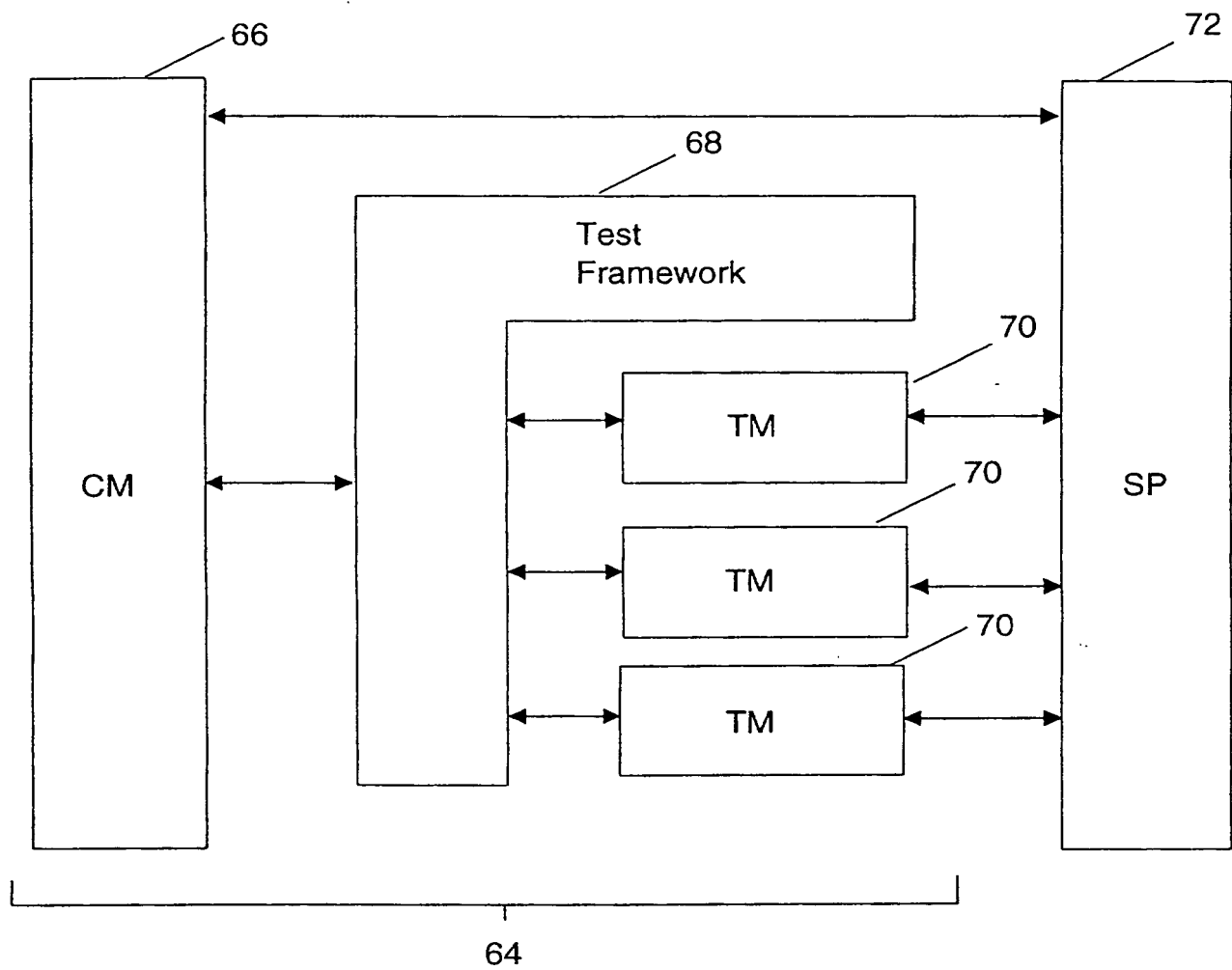
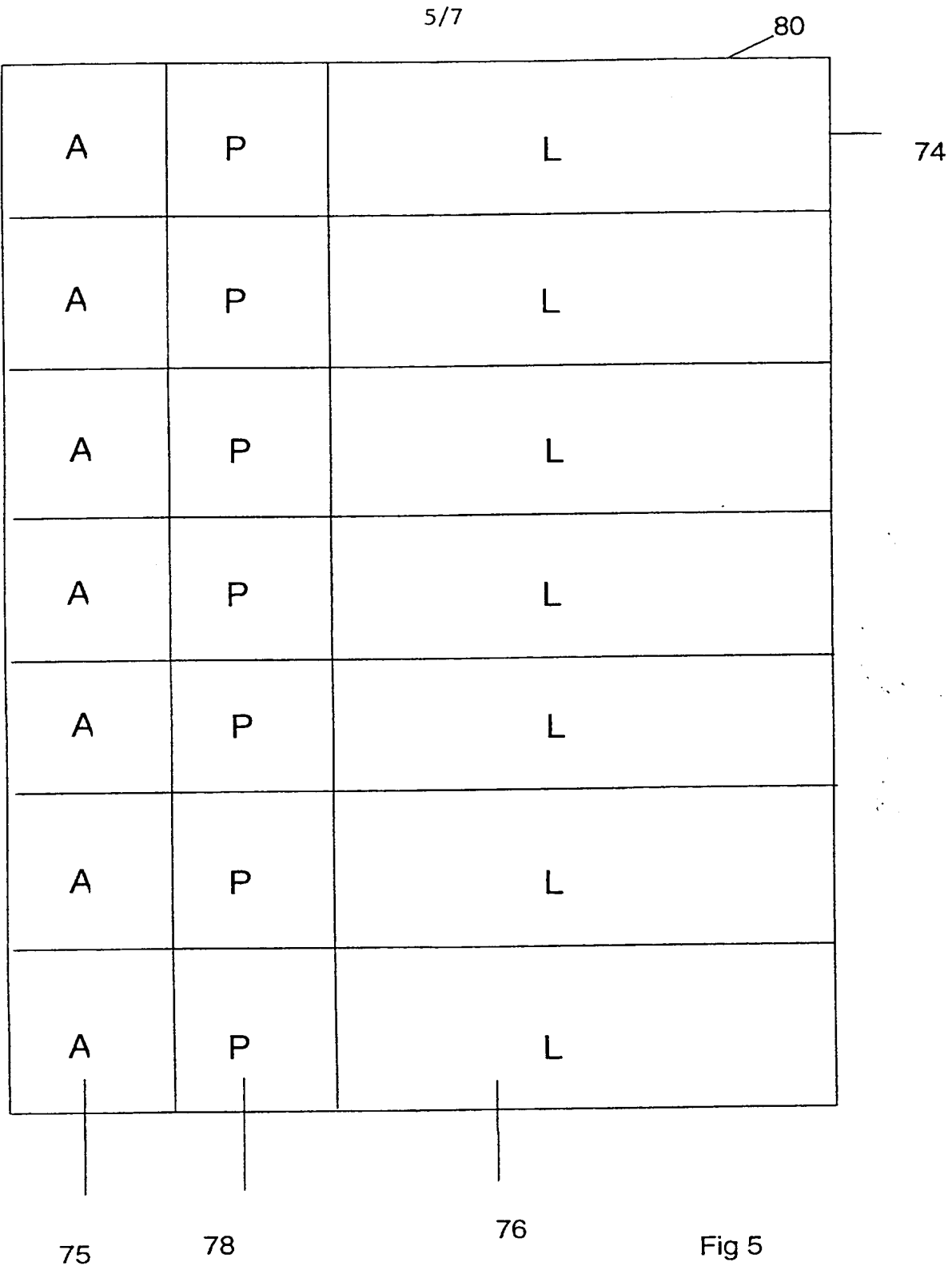


Fig. 4



6/7

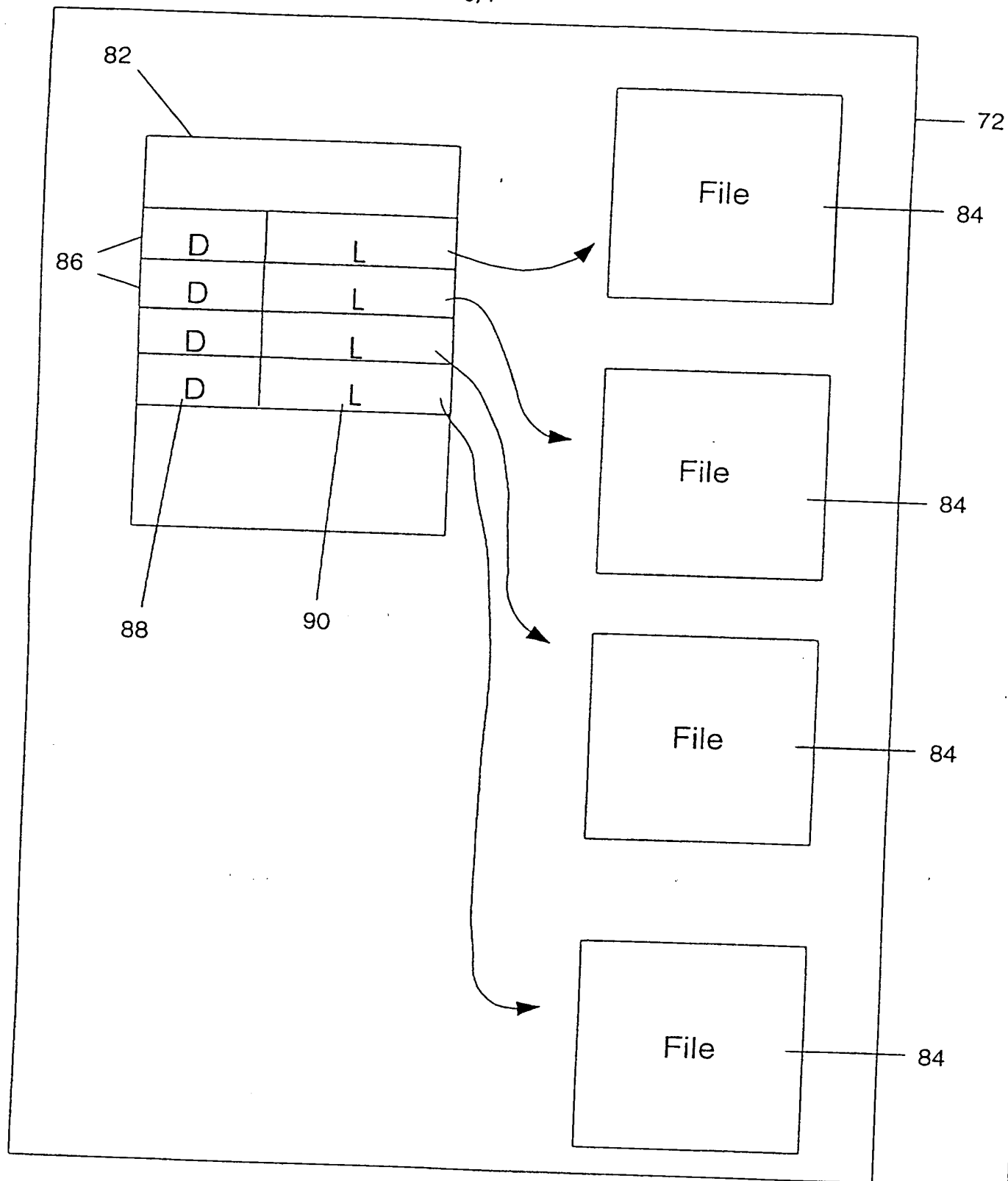


Fig. 6

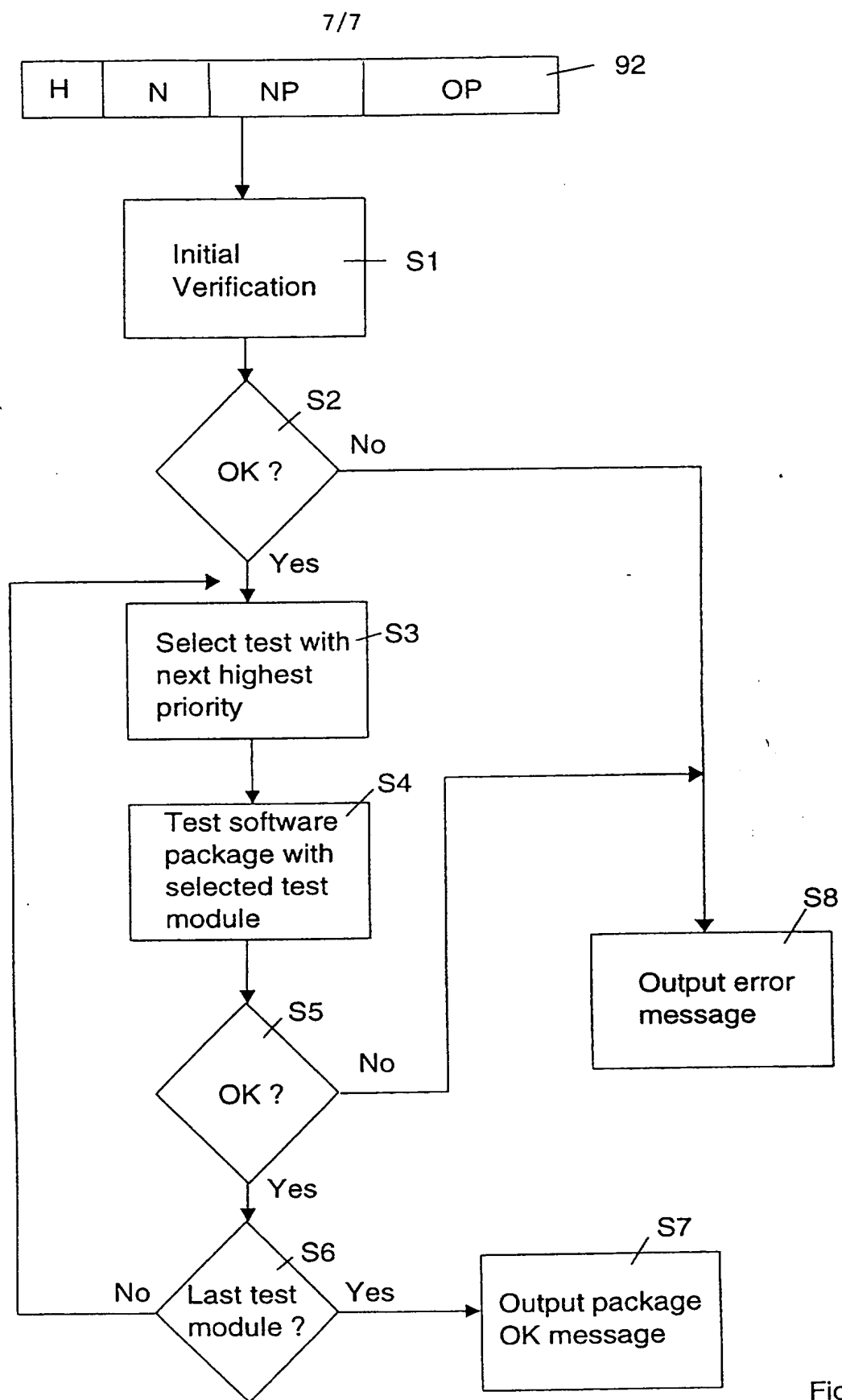


Fig. 7

**THIS PAGE BLANK (USPTO)**